Title:	Count-Min Sketch
Name:	Graham Cormode[1]
Affil./Addr.	Department of Computer Science, University of Warwick, Coventry, UK
Keywords:	streaming algorithms; frequent items; approximate counting, sketch
SumOriWork:	2004; Cormode, Muthukrishnan

Count-Min Sketch

GRAHAM CORMODE[1]

Department of Computer Science, University of Warwick, Coventry, UK

Years aud Authors of Summarized Original Work

2004; Cormode, Muthukrishnan

Keywords

streaming algorithms; frequent items; approximate counting, sketch

Problem Definition

The problem of *sketching* a large mathematical object is to produce a compact data structure that approximately represents it. Much work has focused on the problem of sketching large vectors to provide a small "sketch" of the vector from which key properties – such as the norm of the vector, or estimates of entries – can be retrieved.

The Count-Min (CM) Sketch is an example of a sketch that allows a number of related quantities to be estimated with accuracy guarantees, including point queries and dot product queries. Such queries are at the core of many computations, so the structure can be used in order to answer a variety of other queries, such as frequent items (heavy hitters), quantile finding, join size estimation, and more. Since the sketch can process updates in the form of additions or subtractions to dimensions of the vector (which may correspond to insertions or deletions, or other transactions), it is capable of working over streams of updates, at high rates.

The data structure maintains the linear projection of the vector with a number of other random vectors. These vectors are defined implicitly by simple hash functions. Increasing the range of the hash functions increases the accuracy of the summary, and increasing the number of hash functions decreases the probability of a bad estimate. These tradeoffs are quantified precisely below. Because of this linearity, CM sketches can be scaled, added and subtracted, to produce summaries of the corresponding scaled and combined vectors.

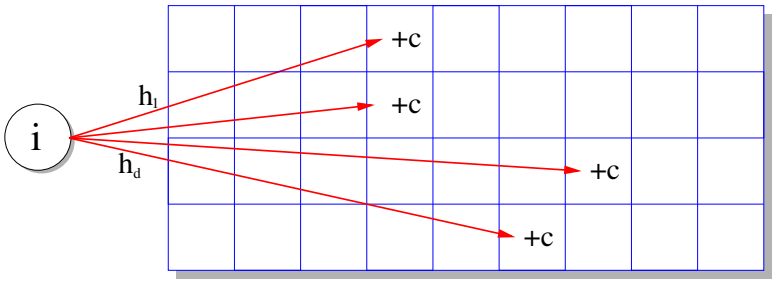

Fig. 1. Each item i is mapped to one cell in each row of the array of counts: when an update of c_t to item i_t arrives, c_t is added to each of these cells

Key Results

The Count-Min sketch was first proposed in 2003 [4], following several other sketch techniques, such as the Count sketch [2] and the AMS sketch [1]. The sketch is similar to a counting Bloom filter or Multistage-Filter [7].

Data Structure Description

The CM sketch is simply an array of counters of width w and depth d, $CM[1,1] \ldots CM[d,w]$. Each entry of the array is initially zero. Additionally, d hash functions

$$h_1 \ldots h_d : \{1 \ldots n\} \rightarrow \{1 \ldots w\}$$

are chosen uniformly at random from a pairwise-independent family. Once w and d are chosen, the space required is fixed: the data structure is represented by wd counters and d hash functions (which can each be represented in $O(1)$ machine words [12]).

Update Procedure. A vector $\boldsymbol{a}$ of dimension n is decscribed incrementally. Initially, $\boldsymbol{a}(0)$ is the zero vector, $\mathbf{0}$, so $a_i(0)$ is 0 for all i. Its state at time t is denoted $\boldsymbol{a}(t) = [a_1(t), \ldots a_i(t), \ldots a_n(t)]$. Updates to individual entries of the vector are presented as a stream of pairs. The tth update is (i_t, c_t), meaning that

$$\begin{aligned} a_{i_t}(t) &= a_{i_t}(t-1) + c_t \\ a_{i'}(t) &= a_{i'}(t-1) \qquad i' \neq i_t \end{aligned}$$

For convenience, the subscript t is dropped, and the current state of the vector simply referred to as $\boldsymbol{a}$. For simplicity of description, it is assumed here that although values of a_i increase and decrease with updates, each $a_i \geq 0$. However, the sketch can also be applied to the case where a_is can be less than zero with some increase in costs [4].

When an update (i_t, c_t) arrives, c_t is added to one count in each row of the Count-Min sketch; the counter is determined by h_j. Formally, given (i_t, c_t), the following modifications are performed:

$$\forall 1 \leq j \leq d : CM[j, h_j(i_t)] \leftarrow CM[j, h_j(i_t)] + c_t$$

This procedure is illustrated in Figure 1. Because computing each hash function takes constant time, the total time to perform an update is $O(d)$, independent of w. Since d is typically small in practice (often less than 10), updates can be processed at high speed.

Point Queries. A *point query* is to estimate the value of an entry in the vector a_i. Given a query point i, an estimate is found from the sketch as $\hat{a}_i = \min_{1 \le j \le d} CM[j, h_j(i)]$. The approximation guarantee is that if $w = \lceil \frac{e}{\varepsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$, the estimate $\hat{a}_i$ obeys $a_i \le \hat{a}_i$; and, with probability at least $1 - \delta$,

$$\hat{a}_i \le a_i + \varepsilon \|\boldsymbol{a}\|_1.$$

Here, $\|\boldsymbol{a}\|_1$ is the L_1 norm of $\boldsymbol{a}$, i.e. the sum of the (absolute) values. The proof follows by using the Markov inequality to bound the error in each row, then using the independence of the hash functions to amplify the success probability.

This analysis makes no assumption about the distribution of values in $\boldsymbol{a}$. In many applications there are Zipfian, or power law, distributions of item frequencies. Here, the (relative) frequency of the ith most frequent item is proportional to i^{-z}, for some parameter z, where z is typically in the range 1—3. Here, the skew in the distribution can be used to show a stronger space/accuracy tradeoff: for a Zipf distribution with parameter z, the space required to answer point queries with error $\varepsilon \|\boldsymbol{a}\|_1$ with probability at least $1 - \delta$ is given by $O(\varepsilon^{-\min\{1,1/z\}} \ln 1/\delta)$ [5].

Range, Heavy Hitter and Quantile Queries. A *range query* is to estimate $\sum_{i=l}^{r} a_i$ for a range $[l \ldots r]$. For small ranges, the range sum can be estimated as a sum of point queries; however, as the range grows, the error in this approach also grows linearly. Instead, $\log n$ sketches can be kept, each of which summarizes a derived vector a^k where

$$a^k[j] = \sum_{i=j2^k}^{(j+1)2^k - 1} a_i$$

for $k = 1 \ldots \log n$. A range of the form $j2^k \ldots (j+1)2^k - 1$ is called a *dyadic range*, and any arbitrary range $[l \ldots r]$ can be partitioned into at most $2 \log n$ dyadic ranges. With appropriate rescaling of accuracy bounds, it follows that Count-Min sketches can be used to find an estimate $\hat{r}$ for a range query on $l \ldots r$ such that

$$\hat{r} - \varepsilon \|\boldsymbol{a}\|_1 \le \sum_{i=l}^{r} a_i \le \hat{r}$$

The right inequality holds with certainty, and the left inequality holds with probability at least $1 - \delta$. The total space required is $O(\frac{\log^2 n}{\varepsilon} \log \frac{1}{\delta})$ [4]. The closely related *ϕ-quantile query* is to find a point j such that

$$\sum_{i=1}^{j} a_i \le \phi \|\boldsymbol{a}\|_1 \le \sum_{i=1}^{j+1} a_i.$$

Range queries can be used to (binary) search for a j which satisfies this requirement approximately (i.e. tolerates up to $\varepsilon \|\boldsymbol{a}\|_1$ error in the above expression) given ϕ. The overall cost is space that depends on $1/\varepsilon$, with further log factors for the rescaling necessary to give the overall guarantee [4]. The time for each insert or delete operation, and the time time to find any quantile, is logarithmic in n, the size of the domain.

Heavy Hitters are those points i such that $a_i \ge \phi \|\boldsymbol{a}\|_1$ for some specified ϕ. The range query primitive based on Count-Min sketches can again be used to find heavy hitters, by recursively splitting dyadic ranges into two and querying each half to see if the range is still heavy, until a range of a single, heavy, item is found. The cost of this is similar to that for quantiles, with space dependent on $1/\varepsilon$ and $\log n$. The time to update the data structure,

and to find approximate heavy hitters, is also logarithmic in n. The guarantee is that every item with frequency at least $(\phi+\varepsilon)\|\boldsymbol{a}\|_1$ is output, and with probability $1-\delta$ no item whose frequency is less than $\phi\|\boldsymbol{a}\|_1$ is output.

Inner product queries. The Count-Min sketch can also be used to estimate the inner product between two vectors. The inner product $\boldsymbol{a}\cdot\boldsymbol{b}$ can be estimated by treating the Count-Min sketch as a collection of d vectors of length w, and finding the minimum inner product between corresponding rows of sketches of the two vectors. With probability $1-\delta$, this estimate is at most an additive quantity $\varepsilon\|\boldsymbol{a}\|_1\|\boldsymbol{b}\|_1$ above the true value of $\boldsymbol{a}\cdot\boldsymbol{b}$. This is to be compared with AMS sketches which guarantee $\varepsilon\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2$ additive error, but require space proportional to $\frac{1}{\varepsilon^2}$ to make this guarantee.

Conservative update. If only positive updates arrive, then the "conservative update" process (due to Estan and Varghese [7]) can be used. For an update (i,c), $\hat{a}_i$ is computed, and the counts are modified according to $\forall 1\leq j\leq d : CM[j,h_j(i)]\leftarrow\max(CM[j,h_j(i)],\hat{a}_i+c)$. This procedure still ensures for point queries that $a_i\leq\hat{a}_i$, and that the error is no worse than in the normal update procedure; it has been observed that conservative update can improve accuracy "up to an order of magnitude" [7]. However, deletions or negative updates can no longer be processed, and the new update procedure is slower than the original one.

Applications

The Count-Min sketch has found a number of applications.

- Indyk [9] used the Count-Min Sketch to estimate the residual mass after removing a set of items. That is, given a (small) set of indices I, to estimate $\sum_{i\notin I}a_i$. This supports clustering over streaming data.
- The *entropy* of a data stream is a function of the relative frequencies of each item or character withn the stream. Using Count-Min Sketches within a larger data structure based on additional hashing techniques, B. Laksminath and Ganguly [8] showed how to estimate this entropy to within relative error.
- Sarlós *et al.* [14] gave approximate algorithms for personalized page rank computations which make use of Count-Min Sketches to compactly represent web-size graphs.
- In describing a system for building selectivity estimates for complex queries, Spiegel and Polyzotis [15] use Count-Min Sketches in order to allow clustering over a high-dimensional space.
- Sketches that reduce the amount of information stored seem like a natural candidate to preserve privacy of information. However, proving privacy requires more care. Roughan and Zhang use the Count-Min sketch to allow private computation of a sketch of a vector [13]. Dwork *et al.* show that the Count-Min sketch can be made *pan-private*, meaning that information about individuals contributing to the data structure is held private.

Experimental Results

There have been a number of experimental studies of CountMin and related algorithms, for a variety of computing models. These have shown that the algorithm is accurate and fast to execute [3; 11]. Implementations on desktop machines achieve between many millions of updates per second, primarily limited by IO throughput. Other implementation have incorporated Count-Min Sketch into high speed streaming systems such as Gigascope [6], and tuned

it to process packet streams of multi-gigabit speeds. Lai and Byrd report on an implementation of Count-Min sketches on a low-power stream processor [10], capable of processing 40 byte packets at a throughput rate of up to 13Gbps. This is equivalent to about 44 million updates per second.

URLs to Code and Data Sets

Sample implementations are widely available in a variety of languages.

C code is given by the MassDal code bank: `http://www.cs.rutgers.edu/~muthu/massdal-code-index.html`.
C++ code due to Marios Hadjieleftheriou is available from `http://research.att.com/~marioh/sketches/index.html`.
The MADlib project has SQL implementations for Postgres/Greenplum `http://madlib.net/`
OCaml implementation is available via `https://github.com/ezyang/ocaml-cminsketch`

Cross-References

AMS Sketch

Recommended Reading

1. Alon N, Matias Y, Szegedy M (1996) The space complexity of approximating the frequency moments. In: ACM Symposium on Theory of Computing, pp 20–29
2. Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)
3. Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Communications of the ACM 52(10):97–105
4. Cormode G, Muthukrishnan S (2005) An improved data stream summary: The Count-Min sketch and its applications. Journal of Algorithms 55(1):58–75
5. Cormode G, Muthukrishnan S (2005) Summarizing and mining skewed data streams. In: SIAM Conference on Data Mining
6. Cormode G, Korn F, Muthukrishnan S, Johnson T, Spatscheck O, Srivastava D (2004) Holistic UDAFs at streaming speeds. In: ACM SIGMOD International Conference on Management of Data, pp 35–46
7. Estan C, Varghese G (2002) New directions in traffic measurement and accounting. In: Proceedings of ACM SIGCOMM, Computer Communication Review, vol 32, 4, pp 323–338
8. Ganguly S, Lakshminath B (2006) Estimating entropy over data streams. In: European Symposium on Algorithms (ESA)
9. Indyk P (2003) Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In: ACM-SIAM Symposium on Discrete Algorithms
10. Lai YK, Byrd GT (2006) High-throughput sketch update on a low-power stream processor. In: Proceedings of the ACM/IEEE symposium on Architecture for networking and communications systems
11. Manerikar N, Palpanas T (2009) Frequent items in streaming data: An experimental evaluation of the state-of-the-art. Data Knowledge Engineering 68(4):415–430
12. Motwani R, Raghavan P (1995) Randomized Algorithms. Cambridge University Press
13. Roughan M, Zhang Y (2006) Secure distributed data mining and its application in large-scale network measurements. ACM SIGCOMM Computer Communication Review (CCR)
14. Sarlós T, Benzúr A, Csalogány K, Fogaras D, Rácz B (2006) To randomize or not to randomize: space optimal summaries for hyperlink analysis. In: International Conference on World Wide Web (WWW)
15. Spiegel J, Polyzotis N (2006) Graph-based synopses for relational selectivity estimation. In: ACM SIGMOD International Conference on Management of Data